Building a Healthcare Analytics Pipeline From Scratch
My Journey Creating an End-to-End Data Engineering Project

Author: *Justin O. Ajuogu*
Date: *November 2025*
Project Duration: *4 weeks*
Tech Stack: *SQL Server, Python, Power BI*

The Challenge
I wanted to build something that demonstrated real-world data engineering skills—not just follow a tutorial. Healthcare analytics seemed like the perfect fit because it combines technical complexity (ETL pipelines, data quality) with real business impact (patient outcomes, operational efficiency) and compliance requirements (HIPAA).

The question I set out to answer: *How can hospitals use data to reduce Emergency Department wait times and improve patient flow?*

What I Built
I created a complete analytics pipeline that processes Emergency Department patient visit data through three transformation layers (Bronze, Silver, Gold) and delivers insights via an interactive Power BI dashboard.

Key Features:

- 300 synthetic patient records spanning 18 days
- 12 automated data quality validation rules
- HIPAA-compliant de-identification (Safe Harbor method)
- Real-time dashboard with 6 visualizations
- Complete technical documentation

The Pipeline:
Synthetic Data → Bronze Layer (raw) → Silver Layer (cleaned) → Gold Layer (analytics) → Power BI Dashboard

---

The Technical Approach

Medallion Architecture
I used the **medallion architecture pattern** (Bronze → Silver → Gold), which is industry standard for modern data platforms like Databricks and Snowflake.
Why this pattern?

- **Separation of concerns:** Raw data preservation vs. transformation logic
- **Clear data lineage:** Easy to trace where data came from and how it changed

- **Scalability:** Each layer can be optimized independently
- **Audit trail:** Complete history of transformations for compliance

Bronze Layer: Raw Data Preservation

The Bronze layer stores data exactly as it comes from the source—no transformations, no quality checks. This is critical for:

- **Debugging:** If something goes wrong downstream, you can always go back to the original data
- **Compliance:** Healthcare regulations often require preserving raw data
- **Reproducibility:** You can re-run transformations without re-extracting from source systems

What I learned: Always preserve your source data. I initially wanted to jump straight to cleaning, but realized I'd have no way to troubleshoot issues or re-process data if my transformation logic changed.

Silver Layer: Data Quality Engineering

This is where the magic happens. The Silver layer applies 12 validation rules to ensure data quality:

Text Standardization:

- Convert all text to UPPERCASE
- Remove leading/trailing whitespace
- Example: "chest pain" → "CHEST PAIN"

Range Validation:

- Acuity levels must be 1-5 (Emergency Severity Index standard)
- Wait times capped at 300 minutes (5 hours max)
- Length of stay under 24 hours (beyond that likely indicates admission, not ED visit)

Derived Fields:

- Extract hour from timestamp (0-23) for hourly analysis
- Parse day of week for staffing pattern analysis
- Create date-only field for daily aggregations

The Challenge: Handling outliers vs. preserving data integrity. Do you reject impossible values (negative wait times) or cap extreme values (300+ minute waits)? I learned to cap rather than reject—better to flag outliers than lose data.

Gold Layer: Analytics-Ready Views

The Gold layer is where business users (and Power BI) connect. Key principles:

HIPAA Compliance:

- Patient IDs replaced with random GUIDs (no way to link back to individuals)
- Exact timestamps aggregated to hour-level only
- Age groups instead of exact ages
- Zero Protected Health Information (PHI) exposed

Business Logic:

- Calculate if 30-minute wait time target was met
- Determine if ESI-specific targets were achieved (Level 1 = immediate, Level 3 = 30 min)
- Join reference tables for user-friendly labels

What I learned: Privacy isn't an afterthought—it needs to be built into the architecture. The Gold layer is specifically designed so that even if someone gets unauthorized access, there's no PHI to expose.

The Dashboard

I built an interactive Power BI dashboard with 6 core visualizations:

KPI Cards:

- Average wait time: 94 minutes
- Total patients: 300

Charts:

- Line chart showing wait times by hour (identifies peak times)
- Bar chart showing patient volume by acuity level
- Pie chart showing chief complaint distribution
- Donut chart showing 30-minute target achievement

Interactivity: The entire dashboard is cross-filtered—click on any visual and all others automatically filter. For example, clicking "Level 3 - Urgent" in the bar chart updates all other visuals to show only Level 3 patients.

The Challenge: Choosing the right aggregations. Should wait time be Average? Median? Max? I went with Average for simplicity, but in production I'd show multiple metrics (average, median, 90th percentile) to catch outliers.

What I learned: Dashboards aren't just about pretty visuals—they're about enabling decisions. Every chart should answer a specific business question: "When are our peak hours?" "Which complaints have longest waits?" "Are we meeting targets?"

Key Insights Discovered

Even with synthetic data, interesting patterns emerged:

Peak Hours:

- Highest volume between 10 AM - 2 PM (60 visits)
- Average wait time of 94 minutes overall
- Recommendation: Add 2 additional triage nurses during peak hours

Acuity Distribution:

- Level 3 (Urgent) is the largest segment at 33% of visits
- This represents mid-acuity patients who are at risk of long waits
- Opportunity: Create fast-track pathway for this segment

Day-of-Week Patterns:

- Monday has 20% more volume than other weekdays
- Weekend volumes are 15% lower
- Staffing models should reflect these patterns

What I learned: Data tells stories if you know how to listen. The 33% of Level 3 patients flagged a real operational challenge—these patients are urgent enough to need care quickly but not critical enough to jump the line. That's where bottlenecks happen.

Challenges & Solutions

Challenge 1: CSV Import Issues
Problem: SQL Server kept throwing data type mismatch errors when trying to BULK INSERT my CSV file.
What I tried:

- Changed column data types in SQL
- Modified CSV formatting
- Tried different delimiters

What actually worked: Using Python with pandas to clean the data first, then inserting row-by-row with proper type casting. It's slower than BULK INSERT but guarantees data integrity.
Lesson learned: Sometimes the "slow" solution is the right solution. Performance optimization comes after correctness.

Challenge 2: Understanding Medallion Architecture
Problem: Initially thought Bronze → Silver → Gold was overkill for a small dataset. Why not just load clean data directly?
What I learned: The pattern isn't about data volume—it's about maintainability and scalability. When my transformation logic changed (I adjusted outlier thresholds), I could re-run Silver without touching Bronze. That separation of concerns saved me hours of debugging.
Lesson learned: Good architecture pays off even on small projects. It's easier to simplify a well-structured pipeline than to refactor a messy one.

Challenge 3: HIPAA Compliance
Problem: How do you de-identify data while keeping it useful for analysis?
What I learned: HIPAA's Safe Harbor method provides 18 specific identifiers to remove or aggregate. The key is removing exact identifiers (names, dates, IDs) while preserving analytical value (age groups, time ranges, geographic regions).
Implementation: Used SQL's NEWID() function to generate random GUIDs, aggregated timestamps to hour-level, and ensured no geographic data below state level.
Lesson learned: Compliance isn't a blocker to analytics—it's a design constraint that makes you think harder about what data you actually need.

Challenge 4: Data Quality Without Losing Data
Problem: How strict should validation rules be? Too lenient = bad data in dashboard. Too strict = reject legitimate edge cases.
My approach:

- **Hard rejections:** NULL patient IDs, NULL timestamps (these make records unusable)

- **Soft corrections:** Cap outliers at realistic maximums, set invalid values to NULL but keep the record
- **Documentation:** Flag every transformation so someone reviewing the data knows what changed

Lesson learned: Data quality is about trade-offs. Document your decisions so future you (or your team) understands the reasoning.

What I Learned (The Real Lessons)

Technical Skills

SQL:

- Writing complex CASE statements for data quality rules
- Understanding JOINS (especially LEFT JOIN for reference data)
- Creating views vs. physical tables (when to use each)
- Date/time functions (DATEPART, DATENAME, CAST)

Python:

- pandas for data manipulation
- pyodbc for SQL Server connections
- Handling data types and type conversions
- Error handling with try-except blocks

Power BI:

- Connecting SQL Server views
- Choosing appropriate visualizations for different data types
- Implementing cross-filtering and interactivity
- Creating calculated measures (though I mostly used SQL for this)

Data Engineering Patterns:

- Medallion architecture (Bronze/Silver/Gold)
- Separation of DDL vs. DML scripts
- ETL vs. ELT (Extract-Transform-Load vs. Extract-Load-Transform)
- Idempotent transformations (can re-run without side effects)

Soft Skills

Documentation: Before this project, I thought documentation was boring busy work. Now I realize it's how you make your work usable by others. I created:

- Data dictionary (what does each field mean?)
- ETL process flow (how does data move through the pipeline?)
- Source-to-target mappings (what transformations are applied?)
- README with setup instructions

Why it matters: If a hiring manager clones my GitHub repo, they can set it up and run it without asking me a single question. That's the mark of professional work.

Problem Scoping: I learned to define the problem before jumping to solutions. My initial instinct was "build a dashboard," but that's a solution. The actual problem was "hospitals need visibility into ED operations to reduce wait times." The dashboard is just one possible solution.

Iterative Development: I didn't build everything at once. I started with:

1. Generate data (validate it works)
2. Load to Bronze (confirm connection works)
3. Transform to Silver (test one rule at a time)
4. Create Gold (verify HIPAA compliance)
5. Build dashboard (one visual at a time)

Each step validated the previous step worked before moving forward.

If I Were to Do This Again

What I'd keep:

- ✅ Medallion architecture—this pattern scales
- ✅ Comprehensive documentation—made sharing the project easy
- ✅ Synthetic data—no privacy concerns, easy to share publicly
- ✅ Version control (GitHub)—could track every change

What I'd change:

- ⚠️ **Add automated testing:** Write SQL tests to validate transformation logic (e.g., "no acuity level should be > 5")
- ⚠️ **Implement incremental loads:** Currently I truncate and reload—in production I'd load only new/changed records
- ⚠️ **Add stored procedures:** Wrap transformations in procedures for easier automation
- ⚠️ **Include error logging:** Track failed records and transformation errors in a separate table
- ⚠️ **Add CI/CD:** Use GitHub Actions to automatically test SQL scripts on commit

What I'd add (future enhancements):

- Real-time streaming (simulate live ED data with Apache Kafka or Azure Event Hub)
- Predictive modeling (forecast wait times based on current volume)
- Alerting (send notifications when wait times exceed thresholds)
- Multiple dashboards (executive summary vs. operational deep dive)

The Business Impact (If This Were Real)

Based on the analysis of 300 patient visits, here's what I'd recommend to hospital leadership:

Immediate Actions:

1. **Increase staffing 10 AM - 2 PM:** This 4-hour window accounts for 20% of daily volume but likely has disproportionate wait times
2. **Implement fast-track for Level 4-5 patients:** These low-acuity patients (45% of volume) could be seen by nurse practitioners, freeing physicians for urgent cases
3. **Monday surge staffing:** 20% higher volume on Mondays requires flex staff or adjusted schedules

Strategic Initiatives:

1. **Build Level 3 optimization pathway:** The 99 Level 3 patients (33% of volume) are the operational challenge—urgent but not critical. Create standardized protocols to move them efficiently
2. **Predictive scheduling:** Use historical patterns to forecast volume by day/hour and schedule accordingly

3. **Patient communication:** Set expectations based on acuity level and current wait times

Expected Impact:

- 15-20% reduction in average wait times through optimized staffing
- 10% reduction in left-without-being-seen (LWBS) rate
- Improved patient satisfaction scores (directly correlated with wait time perception)

Conclusion

This project taught me that data engineering is more than just writing SQL and Python—it's about:

- Understanding the business problem you're solving
- Building systems that others can maintain
- Balancing data quality with practical constraints
- Documenting your work so others can learn from it

I'm proud of what I built, but more importantly, I'm confident I can replicate this process for real-world healthcare analytics challenges. The skills are transferable: ETL pipeline design, data quality engineering, HIPAA compliance, dashboard development, and technical documentation.

Most valuable lesson: Start with the problem, not the tools. I could have built this with different tools (PostgreSQL instead of SQL Server, Tableau instead of Power BI, AWS instead of local infrastructure), but the methodology stays the same—understand your data, ensure quality, deliver insights, document everything.

Project Links

GitHub Repository: EDWT Analysis Demo

Technologies: SQL Server 2019, Python 3.x, Power BI Desktop

Architecture: Medallion (Bronze/Silver/Gold)

Documentation: Complete data dictionary, ETL process flow, source-to-target mappings

Dashboard: 6 interactive visualizations with cross-filtering

This project represents 4 weeks of learning, building, debugging, and documenting. Every line of code taught me something new about data engineering, and I'm excited to apply these skills to real-world healthcare analytics challenges.